Queue Manager Component Binding

Status of this Memo

This is an initial draft of the interface specification for the SSS Queue Manager component. All components that implement the Queue Manager component to be used within the Scalable Systems Software framework must conform to this specification. Guidance is provided as to what MUST, SHOULD and MAY be implemented. Individual applications may support additional features of the SSSRMAP specification and have custom objects and actions that should be documented in an Application Binding document.

Abstract

The Scalable Systems Queue Manager Component uses the SSSRMAP Wire Protocol and SSSRMAP Message Format as its standard interface. This document describes how the Queue Manager component implements these specifications, which features are supported, what meanings are associated with the fields and values, and which objects, actions and attributes may be understood by the component.

Table of Contents

1. Introduction

2. Overview

This document describes the binding between the Queue Manager component and the SSSRMAP specification. It specifies what properties of the specification are utilized, what the meanings are, and in general how to construct the messages and responses for the component. It enumerates the valid objects, actions and options for all valid requests and responses.

3. Wire Protocol

Scalable Systems Software compatible Queue Manager components MUST use the SSSRMAP Wire Protocol for framing and security.

3.1 The *Envelope* Element

No *Envelope* attributes are required. If the *component* attribute is specified, it MUST have a value of “QueueManager”.

3.2 The *Body* Element

The *Body* element MUST specify the *actor* attribute, the value of which must be the userid of the authenticated (or unauthenticated) requestor.

3.3 Transport Layer

This component uses the TCP/IP transport layer according to the SSSRMAP specification.

3.4 Framing

This component uses the HTTP 1.1 POST protocol as per the SSSRMAP specification. For the reply, the Connection property must be specified as “close” since persistent connections are not yet supported.

The message and reply payloads are single XML documents having a root element of *Envelope*.

3.5 Asynchrony

Asynchrony is not yet supported by the Queue Manager component.

3.6 Security

3.6.1 Security Tokens

Conforming components MUST support Symmetric security tokens.

3.7 Authentication

Authentication MUST be supported by a conforming component.

3.8 Privacy

Encryption SHOULD be supported by a conforming component.

4. Message Format

Scalable Systems Software compatible Queue Manager components MUST use the SSSRMAP Message Format for encoding and should be able to validate against the SSSRMAP schema http://www.scidac.org/ScalableSystems/SSSRMAP sssrmap.xsd. The data representation follows the specification's conventions for capitalization and naming.

4.1 Requests and Responses

The *Request* element MUST specify the *object* and *action* attributes. The *Response* elements MAY include these attributes but it is not necessary. The *id* attribute is not used since asynchrony is not yet supported by any implementation.

4.2 Valid Objects

There are two valid objects for the Queue Manager. They are the Queue object and the Job object. Almost all actions apply to the Job object that in many cases may contain nothing more than a JobId.

4.2.1 The *Job* Object

The *Job* object may be the target of all supported actions. The full *Job* object is defined in the SSS Job Object Specification. The Queue Manager utilizes the full *Job* object for the Submit and Query actions. For the remaining actions the Job object need only contain a JobId.

4.2.2 The *Queue* Object

The *Queue* object is an object to provide information about queues. The only valid action for this object is the *Query* action.

4.3 Valid Actions

Valid actions for a particular object are listed in the Object Tables. Most, if not all, objects will support a *Query* action. Most objects will also support *Create*, *Modify*, and *Delete* actions. Some post-only objects will support *Post* instead of *Create*, and may not support *Modify* or *Delete*. Some objects will support additional actions.

4.3.1 The *Submit* Action

The *Submit* action allows for the creation of new jobs. A Job object must be specified in the Data element. The Job element must contain at least the minimum information to specify a job (Actor, Executable, etc).

4.3.2 The *Query* Action

The *Query* action allows for the querying of any set of attributes from any expressible set of objects. *Get*, *Where* and *Option* elements are allowed as content within *Query* requests. One may specify any subset of the attributes as the fields to be returned in any order. If no *Get* elements are included within the request, a default set of fields (possibly all -- but left up to the implementation) will be returned. The *Where* elements are used as the conditions to select the objects for which fields will be returned. Any of the object attributes may be used as the basis for this selection. The reply will contain the results as a series of Job objects in the Data element.

4.3.3 The *Start* Action

The *Start* action attempts to begin the execution of a job. The JobId and the host list for execution must be specified in a Job object within the Data element. The requested JobId must be in the idle state and the number of hosts specified must match the number requested by the job.

4.3.4 The *Cancel* Action

The *Cancel* action requests that the Job specified by the Job object in the Data element be changed to a completed state (Terminated or Completed). This implies that the job must be terminated if currently running and checkpoint data discarded for suspended or checkpointed jobs.

4.3.5 The *Signal* Action

The *Signal* action is used to send a process signal to a job's running processes. The signal to send is specified by a Set element (name="Signal") and the Job is specified by a Job object in the Data element

4.3.6 The *Suspend* Action

The *Suspend* action requests that the job specified in the Job Object in the Data element be suspended to be later resumed on the same node set. This action requires the presence of a CheckPoint Manager. Suspended jobs are resumed with the *Resume* action.

4.3.7 The *Resume* Action

The *Resume* action requests that the job specified in the Job Object in the Data element be resumed. Clearly the requested job must be in the suspended state. This action requires the presence of a CheckPoint Manager.

4.3.8 The *Checkpoint* Action

The *Checkpoint* action requests that the job specified in the Job Object in the Data element be Checkpointed. This action will generate a checkpoint capable of being restarted on a different set of nodes (subject to restrictions within the Checkpoint Manager). To restart a checkpointed job issue a normal *Start* action with the new set of nodes. This action requires the presence of a CheckPoint Manager.

4.3.9 The *Modify* Action

The *Modify* action modifies elements of a job normally specified in the Submit request. This action requires a Job object in the Data element containing the JobId and the elements to modify.

4.3.10 The *Requeue* Action

The *Requeue* action returns a formerly running job back to the idle state to reattempt execution in the future.

4.4 The *Get* Element

The *Get* elements indicate the fields to be returned in a query. Both the *name* and the *op* attribute are supported. Specification of the *name* attribute is REQUIRED. Use of the op element is OPTIONAL and accepts the operation types listed in SSSRMAP. The *units* attribute is not supported.

4.5 The *Set* Element

Set elements are currently used only as specified in the Signal Action.

4.6 The *Where* Element

The *Where* elements are used in conjunction with a *Query* action to specify the set of jobs to query. The Where element MUST contain *name* and *value* attributes. The *name* attribute can be either “JobId” or “JobClass”. In the case of “JobId” the value must contain the job identifier for a valid job. In the case of “JobClass” the value must contain either “all” or “active”. The “active” value will restrict the returned

set of jobs to those with a non-completed status (i.e. queued, running, etc). Multiple Where elements can be used to build a list of jobs.

4.7 The *Status* Element

The Status element is required in all responses.

4.8 The *Code* Element

The Code element is required in all responses. The status codes will be set as described in the Error Reporting section.

4.9 The *Count* Element

The *Count* element is RECOMMENDED in all query responses to indicate the number of objects returned in the query. Other actions may use count in a context-specific manner.

4.10 The *Message* Element

The *Message* element is required in a failure response and optional in the case of a successful response. In both cases the message is context-specific to the failure conditions or the action successfully performed.

4.11 The *Data* Element

The *Data* element is used as a container for Job objects. Most requests and the Query response contain partial or full Job objects. In the simplest form the Job object will contain only a JobId element.

5. Object Tables

6. Error Reporting

Conforming components MUST set the Code element using values defined in the SSSRMAP Message Format specifications. All successful responses MUST set the *Code* element content to 000. Failure responses MUST also set the Code element. At a minimum, the *Code* element content can be set to 999, indicating an unknown

failure, although more specific error codes are highly recommended. All failures should be accompanied by a context-specific message indicating the reason for failure. No component-specific codes are defined.